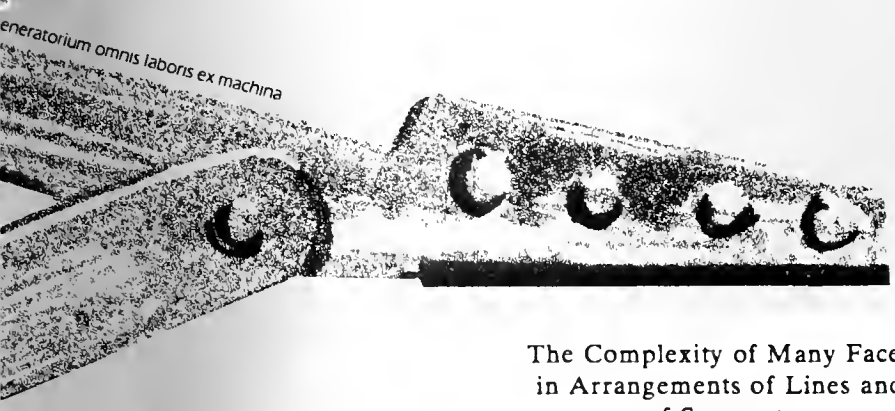
Robotics Research Technical Report

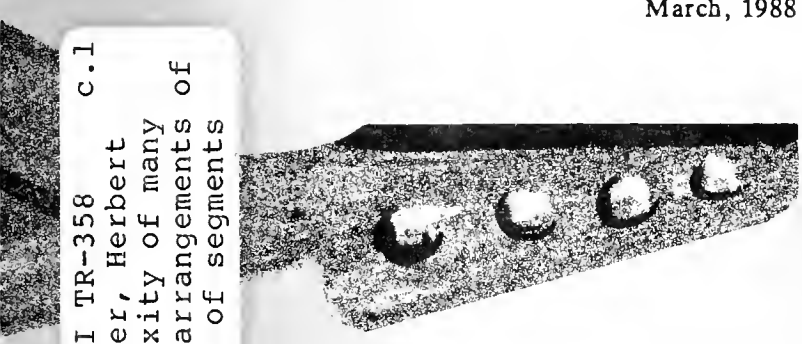


The Complexity of Many Faces in Arrangements of Lines and of Segments

by

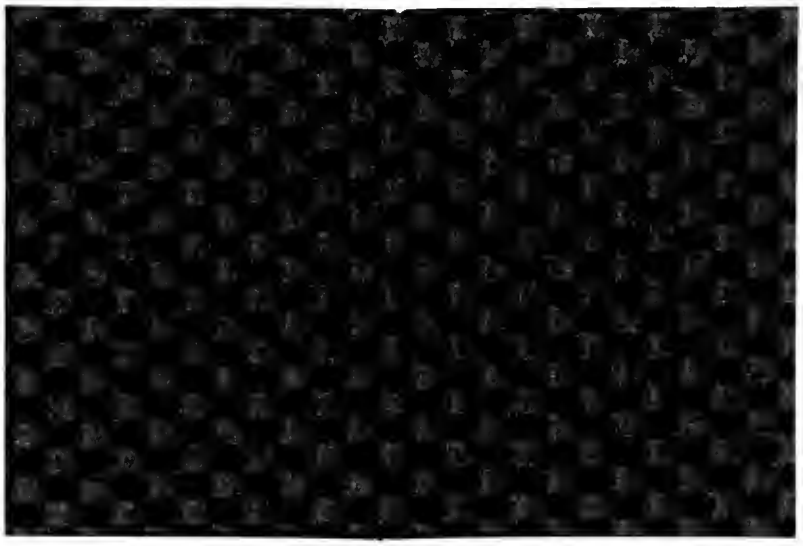
Herbert Edelsbrunner
Leonidas J. Guibas
Micha Sharir

Technical Report No. 358
Robotics Report No. 146
March, 1988



New York University
Courant Institute of Mathematical Sciences

Computer Science Division
251 Mercer Street New York, N.Y. 10012



The Complexity of Many Faces in Arrangements of Lines and of Segments

by

Herbert Edelsbrunner
Leonidas J. Guibas
Micha Sharir

Technical Report No. 358
Robotics Report No. 146
March, 1988

New York University
Dept. of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, New York 10012

The first author is pleased to acknowledge partial support by the Amoco Fnd. Fac. Dev. Comput. Sci. 1-6-44862 and the National Science Foundation under grant CCR-7614575. Work on this paper by the third author has been supported by Office of Naval Research Grant N00014-87-K-0129, by National Science Foundation Grant. No. NSF-DCR-83-20085, by grants from the Digital Equipment Corporation, and the IBM Corporation, and by a research grant from the NCRD — the Israeli National Council for Research and Development.

THE COMPLEXITY OF MANY FACES IN ARRANGEMENTS OF LINES AND OF SEGMENTS[1]

Herbert Edelsbrunner[2], Leonidas J. Guibas[3] and Micha Sharir[4]

Abstract. We show that the total number of edges of m faces of an arrangement of n lines in the plane is $O(m^{2/3-\delta}n^{2/3+2\delta}+n)$, for any $\delta>0$. The proof takes an algorithmic approach, that is, we describe an algorithm for the calculation of these m faces and derive the upper bound from the analysis of the algorithm. The algorithm uses randomization and, with high probability, its time complexity is $O(m^{2/3-\delta}n^{2/3-2\delta}\log n+n\log n\log m)$. If instead of lines we have an arrangement of n line segments, then the maximum number of edges of m faces is $O(m^{2/3-\delta}n^{2/3+2\delta}+n\alpha(n)\log m)$, for any $\delta>0$, where $\alpha(n)$ is the functional inverse of Ackermann's function. We give a (randomized) algorithm that produces these faces and, with high probability, takes time $O(m^{2/3-\delta}n^{2/3-2\delta}\log n+n\alpha(n)\log^2 n\log m)$.

Keywords: Combinatorial geometry, computational geometry, arrangements of lines and line segments, random sampling, probabilistic counting, divide-and-conquer, partition trees, randomized algorithms.

---

[1]The first author is pleased to acknowledge partial support by the Amoco Fnd. Fac. Dev. Comput. Sci. 1-6-44862 and the National Science Foundation under grant CCR-8714565. Work on this paper by the third author has been supported by Office of Naval Research Grant N00014-82-K-0381, by National Science Foundation Grant No. NSF-DCR-83-20085, by grants from the Digital Equipment Corporation, and the IBM Corporation, and by a research grant from the NCRD - the Israeli National Council for Research and Development.

[2]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA.

[3]DEC Systems Research Center, Palo Alto, California 94301, and Department of Computer Science, Stanford University, California 94305, USA.

[4]Courant Institute of Mathematical Sciences, New York University, New York, New York 10012, USA, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

1. Introduction

Let $L=\{l_1,l_2,\ldots,l_n\}$ be a finite set of lines in the plane. L induces a partition of the plane, known as the *arrangement* $A(L)$ of L, into $O(n^2)$ faces, edges, and vertices. The *vertices* are the points of intersection of the lines in L, the *edges* are the connected components of the lines after removing the vertices, and the *faces* are the (convex) connected components of the complement of the union of the lines l_i (see [Gr] or [Ed] for more details concerning arrangements in the plane and in higher dimensions).

Many combinatorial properties of arrangements of lines have been studied extensively. In this paper we consider the maximum number, $K(m,n)$, of edges bounding m distinct faces in an arrangement of n lines in the plane (where we count an edge twice if it bounds two of these faces). Note that m can vary between 1 and $\kappa(n)=\binom{n}{2}+n+1$, and that at these extreme values we have $K(1,n)=n$ and $K(\kappa(n),n)=2n^2$ (there are altogether n^2 edges in the arrangement and each edge bounds two faces). A trivial upper bound for $K(m,n)$ is mn and a trivial lower bound is m. Prior to this and a companion paper [CEGSW], the best known bounds on $K(m,n)$ for general values of m were

(i) $K(m,n)=n+4\binom{m}{2}$ for $m\geq 2$ and $n\geq 4\binom{m}{2}$ [Ca],
(ii) $K(m,n)=O(mn^{1/2})$ for $cn^{1/2}\leq m$ [EW],
(iii) $K(m,n)=O(m^{1/2}n)$ [EW], and
(iv) $K(m,n)=\Omega(m^{2/3}n^{2/3})$ [EW]

(see also [Ed, chapter 6]). Note that each of the upper bounds has a different range of values of m for which it is better than the other (or the trivial) bounds. A graph showing these upper and lower bounds on a logarithmic scale is given in Figure 1.1.

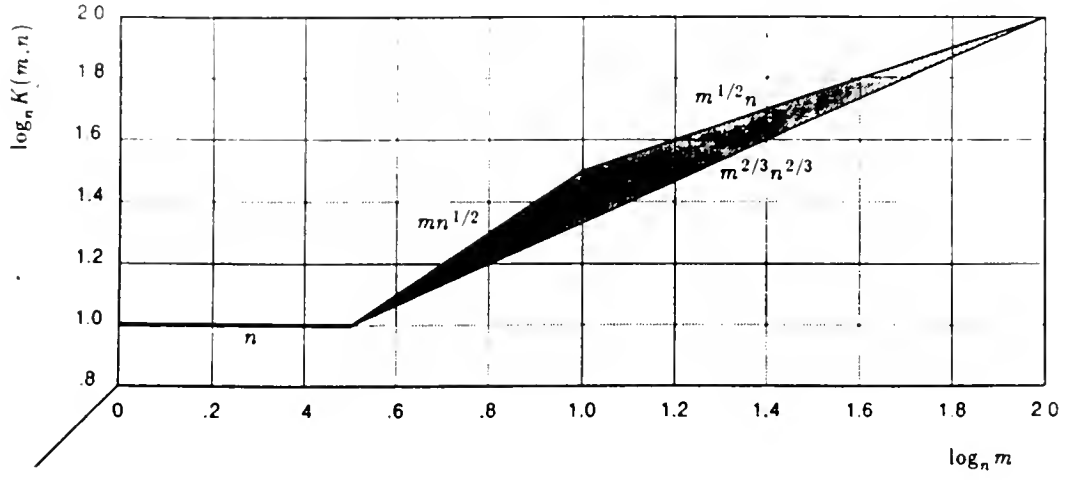


Figure 1.1. Previous bounds on $K(m,n)$.

In this paper we improve the upper bounds by showing that

$$K(m,n) = O(m^{2/3-\delta}n^{2/3-2\delta} + n)$$

for any positive δ (with the constant of proportionality depending on δ)[5]. In particular, when $m = n$ we obtain

$$K(n,n) = O(n^{4/3+\delta})$$

for any $\delta > 0$. Our bound almost matches the lower bound $K(m,n) = \Omega(m^{2/3}n^{2/3})$ obtained by Edelsbrunner and Welzl [EW]. We mention that our upper bound is almost the same as the upper bound due to Szemerédi and Trotter [ST] on the maximum number of incidences between m points and n lines.

Our approach to the combinatorial problem is different from previous work on this problem in that it has an algorithmic flavor. We obtain an algorithm for the calculation of m faces in an arrangement of n lines, where each face is designated by specifying an arbitrary interior point in it. In other words, we consider n lines, $l_1, l_2, \ldots, l_n$, and m points, $p_1, p_2, \ldots, p_m$, in the plane, and calculate the faces of the arrangement that contain the given points (see Figure 1.2; for reasons that will become clear later we allow more than one point designating a single face). We construct these faces using the following divide-and-conquer strategy which mimics the construction of and search in a so-called partition tree which is a data structure designed for half-plane or triangle range queries (see [EW2] and [HW]). It will be convenient to describe this tree in dual space although it is possible to find a fairly natural interpretation of it also in primal space. For this reason, we dualize the points and lines and thus obtain lines p_1^* and points l_1^* in the dual plane. Those lines will be referred to as *dual* lines and the points will be called *dual* points. The tree that we construct can be interpreted in two different ways. Thinking of the dual

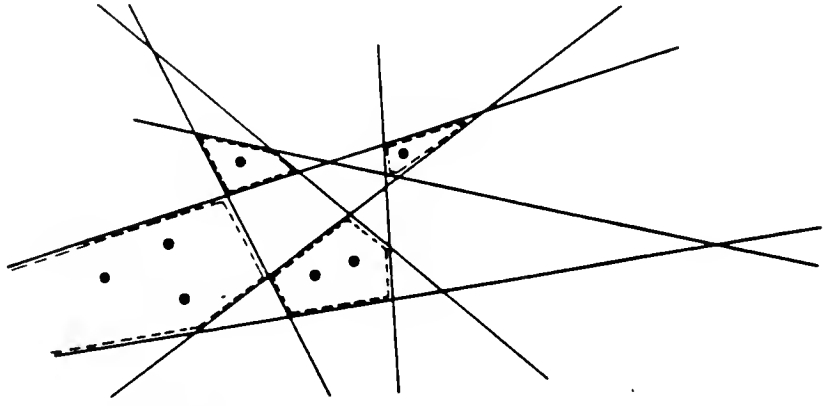


Figure 1.2. Points designate desired faces.

[5]With some effort, one can determine for each m and n the optimal choice of δ, and thus obtain a somewhat tighter bound for $K(m,n)$.

points as data and the dual lines as queries, we obtain a partition of the set of points into disjoint subsets, according to some underlying convex decomposition of the plane; this is similar to standard partition trees [EW2], [HW], except that we use the a priori given query dual lines to form the partition, thus making the tree "customized" and easier to search. An alternative point of view is to think of the dual lines as the data and the dual points as the query objects. In this interpretation, the partition tree that we build differs from standard partition trees in two important aspects — it uses superlinear space since it duplicates data, and is again "customized" for the query objects (the dual points). In particular, we use a stopping rule based on the relation between the numbers of points and lines at any node which decides whether or not the tree is continued below this node. Whichever point of view we take, it is important to keep in mind that both data and queries are available in advance, and that the tree is a function of both. Since the tree may have superlinear size, we will not attempt to really construct it but rather traverse it and build and destroy its nodes as we go. Indeed, the partition tree can be seen as a materialization of the algorithm and exists only on a conceptual level.

Of course, what we want to achieve is the calculation of the faces containing each of the given points, rather than processing half-plane range queries. However, the two problems turn out to have a lot in common, so that they are both amenable to the partition tree technique. Nevertheless, the two problems require different actions when it comes to combining information from the children of a node to produce the output at that node. This is a fairly trivial step in range searching but requires some sophisticated machinery in our case. Specifically, for a node v of our tree and for each (primal) point p that reaches that node, we want to construct the face in the arrangement of the (primal) lines that reach v. This is accomplished by combining the faces containing p in each of the subarrangements corresponding to the children of v. A major tool that we develop for this purpose is the so-called "combination lemma" which gives a tight upper bound on the maximum combinatorial complexity[6] of the desired faces in terms of the combinatorial complexity of the corresponding faces in the subarrangements (see Lemma 1). We expect this result to have applications to other problems as well.

To re-iterate, we present a technique for constructing a partition tree for a set of "data" points and a predetermined set of "query" lines. Such a tree can then be used

(a) to obtain better bounds for batched half-plane range searching when the queries are known in advance (applications include calculating the "signature" of a polygonal curve [OR], multiple ray-tracing [SML], etc.),

(b) to obtain our bounds on the complexity of many faces in arrangements of lines (or of line segments, as studied in Sections 4 through 7), and

(c) in many other applications of a similar nature, such as reporting or

[6]We use the term "combinatorial complexity" and sometimes just "complexity" for the number of edges bounding some collection of faces.

counting the intersections between n given line segments (see [GOS]).

In our present application, the desired upper bound on $K(m,n)$ is obtained by analyzing the space complexity of the resulting algorithm. The time complexity of the algorithm is roughly a polylogarithmic factor times the upper bound on $K(m,n)$ mentioned above (see Section 3 for a more precise bound). The algorithm is based on a random sampling technique akin to the ϵ-net method of Haussler and Welzl [HW] and to the random sampling method of Clarkson [Cl]. We obtain a randomized algorithm which always terminates and produces the desired output and which, with high probability, does so within the stated time bound.

Next we consider the problem of estimating the maximum number of edges bounding m faces in an arrangement A of n line segments in the plane, and of calculating these faces. This problem is considerably more difficult than for lines, because the faces of A are not necessarily convex or simply connected. This makes it harder to process such faces efficiently. Nevertheless, using an intricate extension of our combination lemma (see Lemma 5), we obtain essentially the same bound on the maximum complexity, $R(m,n)$, of m distinct faces in an arrangement of n line segments. More precisely, we prove

$$R(m,n) = O(m^{2/3-\delta}n^{2/3+2\delta} + n\alpha(n)\log m)$$

for any $\delta>0$, where $\alpha(n)$ is the extremely slowly growing inverse of Ackermann's function. To the best of our knowledge this is the first non-trivial upper bound known for $R(m,n)$. Note that this upper bound almost matches the above mentioned lower bound on $K(m,n)$. Since trivially $K(m,n)\leq R(m,n)$ this implies that our upper bound on $R(m,n)$ is almost tight.

From a high-level point of view the algorithms for the calculation of the desired faces in arrangements of lines and of line segments are quite similar. Both algorithms employ a key procedure for the following problem:

> given a collection of k points and the faces containing them in each of two subarrangements of the given lines or line segments, calculate the faces containing these points in the arrangement formed by the union (that is, overlay) of the two subarrangements.

In the case of lines this is easy to achieve efficiently because each face is convex. In the case of line segments this is more difficult because of the potentially highly irregular shapes of individual faces. We present an efficient line sweeping method for merging faces containing k given points in line segment arrangements whose complexity is $O((t+k)\log(t+k))$, where t is the total complexity of input and output faces. Applying this merge recursively, we can calculate the required faces in time which is within a polylogarithmic factor of the bound on $R(m,n)$. An interesting consequence of our merging procedure is that a single face in an arrangement of n line segments in the plane can be constructed in time $O(n\alpha(n)\log^2 n)$. This problem arises in certain two-dimensional motion planning problems in robotics, and has been previously studied in [PSS]. A companion paper, [GSS], extends the line-sweep technique of this paper to

the calculation of a single face in arrangements of more general curves.

The technique used in this paper is one of several related approaches that were developed recently, all of which use ϵ–nets and random sampling as basic tools. This paper uses ϵ–nets to partition the given lines (or line segments) into a fixed number of (disjoint) subsets so that each subset interacts only with a relatively small number of the given points. These interactions are taken care of recursively. In contrast, one might try to partition the given points into (disjoint) subsets, each interacting with only a small number of the given lines (or line segments). This alternative approach has been studied in a companion paper [CEGSW]. It yields tight combinatorial results for the case of lines, and can be used to obtain upper bounds for the complexity of many faces, and for the total number of incidences with many points, in arrangements of other types of curves, and also in arrangements in higher dimensions. We call the approach followed in this paper *dual* while we refer to the approach in [CEGSW] as being *primal*. While the primal approach is mainly combinatorial, the dual method yields efficient randomized algorithms. Another advantage of the dual method over the primal is that it extends to line segments (which have not been amenable to primal investigations yet). In addition, the dual approach has turned out to be better than the primal one in analyzing the complexity of many cells in arrangements of planes or hyperplanes, as is demonstrated in another companion paper [EGSh].

The paper is organized as follows. In Section 2 we analyze the combinatorial complexity of many faces in an arrangement of lines. This analysis is explained in terms of an algorithm that constructs the faces; its implementation is discussed in Section 3. In Sections 4 and 5 we analyze the combinatorial complexity of many faces in an arrangement of line segments, and in Sections 6 and 7 we discuss the implementation of the algorithm implicitly described in the combinatorial analysis. Concluding remarks and open problems are given in Section 8.

2. The Complexity of Many Faces in an Arrangement of Lines

Let $L=\{l_1,l_2,\ldots,l_n\}$ be a set of n lines in the plane, and let $A=A(L)$ denote their arrangement as defined in the introduction. Let $p_1,p_2,\ldots,p_m$ be m given points that do not lie on any of these lines. Consider the problem of calculating all faces of A that contain the points p_i producing each such face just once, even if it contains several of these points (see Figure 1.2). We seek an algorithm for solving this problem with a small worst–case space complexity. This space complexity will serve as an upper bound on the maximum number of edges bounding any m faces in any arrangement of n lines in the plane. As will turn out, the time complexity of our (randomized) algorithm will be, with high probability, within a $\log n$ factor (and a $\log n \log m$ factor if $m=O(\sqrt{n})$) of its worst–case space complexity, so we also get a nearly time–optimal (although randomized) algorithm for the calculation of the faces.

We assume that initially no two of the given points lie in the same face of A.

The algorithm that we present below uses a divide-and-conquer approach and each recursive step involves some subset L' of the lines l_i and some subset P' of the points p_j. Since L' is only a subset of L it thus can happen that in the arrangement formed by L' two or more points of P' fall into the same face. In this case we will want the algorithm to maintain this face just once, and have pointers to it from each of the points contained in it. To reflect this potential duplication, we will denote by $K(m,n)$ the maximum complexity of the faces in an arrangement of n lines that contain m given points (counting each face just once). As opposed to $K(m,n)$ which is defined only if $m \leq \kappa(n)$, $K(m,n)$ is defined for all integers $m \geq 0$, $n > 0$. However, when both functions are defined, we clearly have $K(m,n)=K(m,n)$.

We next describe the algorithm for calculating the required faces. The discussion will ignore implementation issues (addressed in Section 3) and instead concentrate on combinatorial problems that arise.

First we dualize the lines l_i to points l_i^* and the points p_i to lines p_i^*. This gives a set L^* of n points and a set P^* of m lines in the dual plane. To process the dual points and lines, we choose some constant integer $r > 0$, and select a random sample of r of the dual lines p_j^*. When we draw the arrangement of these lines in the dual plane and triangulate each of the faces of this arrangement we obtain a total of $M = O(r^2)$ triangles. The ϵ-net theory of Haussler and Welzl [HW] or, alternatively, the random sampling lemma of Clarkson [Cl] imply that, with high probability, the interior of each of these triangles intersects at most $\frac{cm}{r}\log r$ dual lines, for some constant c. We now build a partition tree, T, as follows. Each node of T is associated with some subset of the dual points of L^* and with some subset of the dual lines of P^*. Let v be a node of T that is associated with $L_v^* \subseteq L^*$ and $P_v^* \subseteq P^*$. The points in L_v^* and the lines in P_v^* (as well as their primal counterparts) are said to *reach* v. We take a random sample of r lines from P_v^* and construct and triangulate their arrangement. For each triangle Δ_w in this arrangement we form a child w of v in T and associate with w the subset L_w^* of the dual points of L_v^* contained in Δ_w and the subset P_w^* of dual lines of P_v^* that intersect the interior of Δ_w. (A schematic representation of this process is shown in Figure 2.1; the arrangement is formed by $r = 2$ lines, thus it is not necessary to further decompose the four faces which correspond to the four children of v.) In what follows we will denote the cardinality of L_v^* by n_v and the cardinality of P_v^* by m_v[7].

This process is continued recursively, but not all the way until just one or no point or line remains. Whenever we reach a node v of T for which $m_v \geq \kappa(n_v)$, we stop the process and undo the dualization to obtain L_v from L_v^* and P_v from P_v^*. Then we construct the arrangement $A(L_v)$ (in the primal plane), locate[8] in it each of

[7]The reader is advised to note that we consistently use the letters L and n in association with the primal lines (and therefore with the dual points) and that P and m are used in connection with primal points (and thus dual lines).

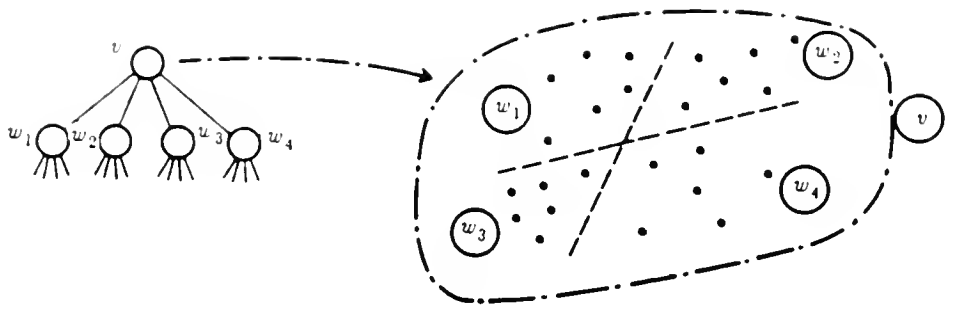


Figure 2.1. Partition tree and corresponding decomposition.

the points of P_v, and report the faces of $A(L_v)$ that contain them. Another case where we stop in the construction of $\mathcal{T}$ at node v is if $P_v^* = \emptyset$. In this case we do not have to bother constructing $A(L_v)$ since there are no points for which faces need to be calculated.

During this process we construct the required collection of faces of $A(L)$ as follows. For each node v of $\mathcal{T}$ and for each point p_i reaching v during the above construction, let $F_v(p_i)$ denote the face of the arrangement $A(L_v)$ that contains p_i. (Recall that such a face will be shared by all points reaching v that lie in it.) These faces are constructed bottom-up as follows.

(i) As mentioned above, at each leaf v of $\mathcal{T}$ we either have $m_v \geq \kappa(n_v)$ or $m_v = 0$. In the first case we construct the entire arrangement of the n_v corresponding lines l_j, locate in this arrangement each of the m_v points p_i of P_v^*, and collect the corresponding faces $F_v(p_i)$ (each face only once). The total number of edges bounding these faces (which is proportional to the space needed to store them) is at most $O(n_v^2) = O(m_v)$. In passing we mention that the time-complexity of this step is at most $O(n_v^2 + m_v \log n_v) = O(m_v \log n_v)$ using the arrangement construction algorithm of [EOS] and the optimal point location structure of [EGSt].

(ii) At each node v of $\mathcal{T}$, compute $Q_v = P_u - P_v$, u the parent of v, and let b_v be the cardinality of Q_v. Q_v is the set of points whose dual lines intersect the interior of Δ_u but miss the interior of Δ_v. By duality, each of these points p_i lies either above all the lines in L_v (in the topmost face F_v^- of $A(L_v)$) or below all of them (in the bottommost face F_v^-). This is illustrated in Figure 2.2. These two faces together have at most $n_v + 2$ edges, and they can be constructed in time $O(n_v \log n_v)$ (see e.g. [PS]). As required, we store each of these two faces just once, and maintain a pointer from each point

[8] *Locating* a point in an arrangement means to find the face (or edge or vertex) of the arrangement that contains the point. It is a fairly common term in computational geometry which, among other things, considers data structures that facilitate fast point location queries (see e.g. [EGSt]).

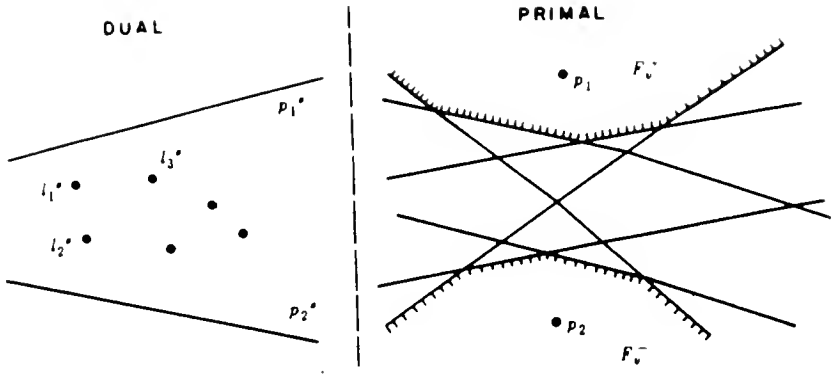


Figure 2.2. Dual and primal plane.

$p_i \in Q_v$ to either F_v^+ or F_v^- whichever contains it. We note that the topmost and bottommost faces in $A(L_v)$ must be constructed even if $m_v = 0$ and we stop the construction of the tree below v.

(iii) The general recursive step at any internal node v of T proceeds as follows. Let $w_1, w_2, \ldots, w_M$ be the children of v (recall that $M = O(r^2)$). For each point p_i that reaches v (that is, its dual line intersects Δ_v and the triangles of all ancestors of v) and for each child w_j of v, either p_i belongs to Q_{w_j} or p_i reaches w_j, in which case the face $F_{w_j}(p_i)$ containing p_i has been calculated recursively. Our task is now to take the M faces containing p_i (where each such face is either of the form $F_{w_j}(p_i)$ or is one of $F_{w_j}^+$, $F_{w_j}^-$ associated with Q_{w_j}) and to form their intersection to obtain $F_v(p_i)$. This intersection is clearly a convex polygon that contains p_i and the number of its edges is at most the sum of the number of edges of the intersected faces.

However, since some of the faces may be shared by other points, we need to avoid duplicate processing and counting of the same face for each point it contains, or else our algorithm might have unacceptably high time-complexity and our upper bound on the total number of edges will be annoyingly loose. A typical case where duplicate processing of this sort can slow down the algorithm is depicted in Figure 2.3.

A solution to this problem is provided by the following technical lemma, which we refer to as the "combination lemma (for lines)".

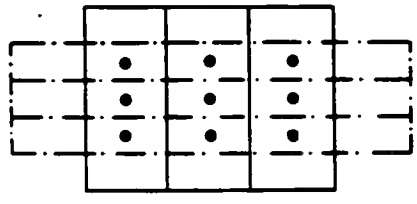


Figure 2.3. Six faces designated by three points each.

Lemma 1. Let $p_1,p_2,\ldots,p_k$ be points in the plane, and let $\{B_1,B_2,\ldots,B_s\}$ and $\{R_1,R_2,\ldots,R_t\}$ be collections of s "blue" and t "red" open convex polygons that satisfy the following three conditions.

(i) The blue (red) polygons are pairwise disjoint and the total number of blue (red) edges is β (ρ).

(ii) Each point p_i is contained in a blue polygon B_{s_i} and in a red polygon R_{t_i}.

(iii) If for each $1 \leq i \leq k$ we define $E_i = B_{s_i} \cap R_{t_i}$, then $E_i \neq E_j$ if $i \neq j$.

Then the total number of sides of the E_i is at most $\beta+\rho+4k-2s-2t$.

Proof. Take one of the blue polygons $B = B_j$, and suppose that it contains k_j points, say $p_1,p_2,\ldots,p_{k_j}$. Each of these points p_i lies in a different red polygon R_{t_i}. We consider the k_j cells $E_i = B \cap R_{t_i}$, for $1 \leq i \leq k_j$, which are convex polygons (see Figure 2.4). To give an upper bound on the number of blue edges of the E_i we define for an edge e of B the intersection of e with R_{t_i} and denote it by e_i. Now write down the cyclic sequence of the non-empty e_i in clockwise order around the boundary, ∂B, of B. We observe the following two properties.

(i) The sequence of indices (red polygons intersecting ∂B) contains no cyclic scattered subsequence of the form $i..j..i..j$.

(ii) If two consecutive indices (red polygons) are the same, then the edges of B in both elements are different.

To prove (i) just note that if such a case were to arise then we could connect the first and third edges and the second and fourth edges by two straight segments lying respectively inside the red convex polygons R_{t_i} and R_{t_j}. Both segments have their endpoints on ∂B which implies by the Jordan curve theorem that they intersect.

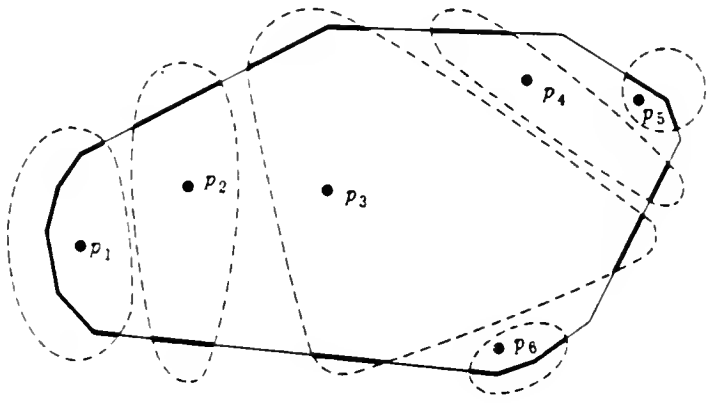


Figure 2.4. One blue and six red faces.

This is a contradiction to $R_{t_i} \cap R_{t_j} = \emptyset$ if $i \neq j$. The claim (ii) follows from the fact that a single edge of B intersects a red polygon in a connected piece – after all both the blue edge and the red polygon are convex.

Ignoring repetitions of indices, (i) implies that the cyclic sequence is a Davenport–Schinzel cycle of order 2 and thus consists of at most $2k_j-2$ edges (see [ES] for details if at all necessary). By (ii), the number of index repetitions is at most $|B|$, the number of edges of B. It follows that the E_i, for $1 \leq i \leq k_j$, have at most $|B|+2k_j-2$ blue edges.

If we take the sum over all blue polygons we get at most $\beta+2k-2s$ blue edges bounding the cells E_i, for all i. By a symmetric argument we can show that the number of red edges bounding the E_i is at most $\rho+2k-2t$. It follows that the total number of edges of the E_i (each edge either blue or red) is at most $\beta+\rho+4k-2s-2t$. □

By applying Lemma 1 a fixed number of times ($M-1$ times to be precise), it follows that the overall complexity, K_v, of all faces $F_v(p_i)$ constructed at v can be bounded from above by

$$\sum_{j=1}^{M}(K_{w_j}+(n_{w_j}+2))+M\sum_{j=1}^{M}4m_{w_j} = \sum_{j=1}^{M}(K_{w_j}+O(n_{w_j}))+M\sum_{j=1}^{M}O(m_{w_j}),$$

where K_{w_j} is the number of edges counted at w_j, and the second subterm of the first sum arises from the faces $F_{w_j}^-$ and $F_{w_j}^-$. But $\sum_j n_{w_j} = n_v$ and $M\sum_{j=1}^{M}O(m_{w_j}) = O(m_v)$, since r and thus M is a constant. We can thus rewrite the recurrence relation as

$$K_v = \sum_{j=1}^{M} K_{w_j} + O(m_v+n_v).$$

To solve this recurrence relation, let $\overline{K}(m,n)$ denote as above the maximum complexity of the collection of faces that arise for m points in an arrangement of n lines. Then we have

$$\overline{K}(m,n) \leq \begin{cases} 0 & \text{if } m=0 \\ am & \text{if } m \geq \kappa(n) \\ \sum_{i=1}^{M}\overline{K}(m_i,n_i)+bm+b'n & \text{if } m < \kappa(n) \end{cases}$$

for some constants $a,b,b'>0$, where M, the m_i, and the n_i satisfy the following three conditions (which are immediate from our construction)

$$M = O(r^2), \tag{I}$$

$$\sum_{i=1}^{M} n_i = n, \text{ and} \tag{II}$$

$$\text{for each } i \text{ we have } m_i \leq \frac{cm}{r}\log r, \text{ for some constant } c>0. \qquad \text{(III)}$$

Under these constraints we have

Lemma 2. $\overline{K}(m,n) \leq Dm^{2/3-\delta}n^{2/3+2\delta}+Am+Bn\log m$, for any $\delta>0$, where the coefficients A,B,D depend on δ.

Proof. We first note that at each level of the recursion m decreases by a factor $\Omega(\frac{\log r}{r})$, for a constant r, and thus the recursion has only $O(\log m)$ levels. The sum of the n_v, over all nodes v at the same recursion level, is clearly n, so that the total contribution of the rightmost term, $b'n$, is at most $O(n\log m)$. We will thus ignore this term in the recurrence relation for $\overline{K}$ and prove that the solution to the modified recurrence satisfies $\overline{K}(m,n) \leq Dm^{2/3-\delta}n^{2/3+2\delta}+Am$, for any $\delta>0$.

Fix $\delta>0$ and choose $r=r(\delta)>0$ sufficiently large (how large will be apparent from the analysis below).

The bound is trivial for $m=0$. If $m \geq \kappa(n)$ then $\overline{K}(m,n) \leq am$ plainly satisfies the required inequality, assuming $A \geq a$. It follows that the bound is trivially true for constant n since $m<\kappa(n)$ only if m is also at most a constant (we need this observation only for $n \leq 1$). So suppose $m<\kappa(n)$. In this case

$$m = m^{2/3-\delta}m^{1/3+\delta} \leq m^{2/3-\delta}n^{2/3+2\delta}, \qquad (*)$$

assuming $n \geq 2$. By induction hypothesis we then have

$$\overline{K}(m,n) \leq \sum_{i=1}^{M}(Dm_i^{2/3-\delta}n_i^{2/3+2\delta}+Am_i) + bm.$$

By properties (III) and (I) we have

$$\sum_{i=1}^{M} m_i \leq \frac{cMm\log r}{r} \leq (c_1 r\log r)m,$$

for some constant c_1. Hence

$$\overline{K}(m,n) \leq D\cdot\sum_{i=1}^{M} m_i^{2/3-\delta}n_i^{2/3+2\delta} + (Ac_1 r\log r + b)m.$$

Thus, using (*) and putting $d=Ac_1 r\log r+b$, we obtain

$$\overline{K}(m,n) \leq D\cdot\sum_{i=1}^{M} m_i^{2/3-\delta}n_i^{2/3+2\delta} + dm^{2/3-\delta}n^{2/3+2\delta}.$$

But

$$\sum_{i=1}^{M} m_i^{2/3-\delta}n_i^{2/3-2\delta} \leq (\frac{cm\log r}{r})^{2/3-\delta}\sum_{i=1}^{M} n_i^{2/3-2\delta}$$

which, by the Hölder–Minkowski inequality, does not exceed

$$\left(\frac{cm\log r}{r}\right)^{2/3-\delta}n^{2/3-2\delta}M^{1/3-2\delta} = O\left(\frac{(\log r)^{2/3-\delta}}{r^{3\delta}}m^{2/3-\delta}n^{2/3-2\delta}\right).$$

Hence

$$\overline{K}(m,n) \le \left(D\cdot O\left(\frac{(\log r)^{2/3-\delta}}{r^{3\delta}}\right)+d\right)m^{2/3-\delta}n^{2/3+2\delta}.$$

But since $\delta>0$, it is clear that if r is chosen sufficiently large so that $O\left(\frac{(\log r)^{2/3-\delta}}{r^{3\delta}}\right)<\frac{1}{2}$, say, and if D is taken to be sufficiently large so that $\frac{D}{2}>d=Ac_1 r\log r+b$, then the expression in the bracket will be less than or equal to D, thus establishing the asserted inequality. □

Theorem 3. The total number of edges, $K(m,n)$, bounding m distinct faces in an arrangement of n lines is at most $O(m^{2/3-\delta}n^{2/3+2\delta}+n)$, for any $\delta>0$ (where the constants of proportionality depend on δ).

Proof. Recall that when both functions are defined, we have $K(m,n)=\overline{K}(m,n)$. For $m\le n^{1/2}$ the asserted bound follows immediately from the results of [Ca] mentioned in the introduction. For $n^{1/2}<m\le\kappa(n)$ it is easily checked that the term $O(m^{2/3-\delta}n^{2/3-2\delta})$ dominates $O(m)$ and $O(n\log m)$ in the bound of Lemma 2. □

Remarks. (1) The preceding bounds imply that $K(m,n)=O(m^{2/3-\delta}n^{2/3-2\delta})$ for any $\delta>0$, provided neither m nor n is too small.

(2) Our result leaves a small gap between our upper bound and the lower bound of $\Omega(m^{2/3}n^{2/3}+n)$ obtained in [EW]. The primal approach as presented in a companion paper [CEGSW] closes this gap and shows that $\Theta(m^{2/3}n^{2/3}+n)$ is the real bound.

(3) A related result is that of Szemerédi and Trotter [ST] who give a tight bound, $\Theta(m^{2/3}n^{2/3}+n)$, on the maximum sum of the degrees of m vertices in an arrangement of n lines. There does not appear to be an easy way to extend the proof technique of [ST] to the case of faces.

3. Calculating Many Faces in an Arrangement of Lines

To complete our analysis of line arrangements, we turn to the implementation of the algorithm outlined in Section 2 which constructs the faces in an arrangement of n lines $l_1,l_2,\ldots,l_n$ that contain m given points $p_1,p_2,\ldots,p_m$. Let $T(m,n)$ denote the time needed for this task using the approach described in Section 2. We have already noted that at the bottom of the recursion we have $T(m,n)=O(m\log n)$ if $m\ge\kappa(n)$ and $T(m,n)=O(n\log n)$ if $m=0$. Furthermore, the calculation of the two faces F_v^- and F_v^- at any node v costs time $O(n_v\log n_v)$.

As for the general merging step at some node v of $\mathcal{T}$, let $p_1,p_2,\ldots,p_{m_v}$ be the

points whose dual lines reach v. For each p_i we need to calculate the face $F_v(p_i)$ of the arrangement $A(L_v)$ that contains p_i: this face is the intersection of the M faces $\tilde{F}_{w_j}(p_i)$, $j=1,2,\ldots,M$, where $\tilde{F}_{w_j}(p_i)$ equals $F_{w_j}(p_i)$ if p_i^* reaches the child w_j of v, and $\tilde{F}_{w_j}(p_i)$ is $F_{w_j}^-$ or $F_{w_j}^-$ if $p_i \in Q_{w_j}$. Recall that a major technical difficulty in the analysis of the space complexity of the algorithm, given in the preceding section, was to avoid duplicate access to a face that is shared by several of the points. To overcome this difficulty algorithmically, we proceed as follows. For expository reasons we assume $M=2$, so that we need to intersect only two faces around each p_i.

(i) With each p_i we associate the pair $(\tilde{F}_{w_1}(p_i), \tilde{F}_{w_2}(p_i))$. Regard two points as equivalent if they have the same associated pair of faces. The equivalence classes can be constructed in time $O(m_v \log m_v) = O(m_v \log n_v)$ by sorting the face-pairs and removing repetitions. This also yields a representative point for each equivalence class; we need to calculate $F_v(p_i)$ only for these representative points.

(ii) For each representative point p_i, we need to calculate the intersection, E, of the two convex polygons, $B=\tilde{F}_{w_1}(p_i)$ and $R=\tilde{F}_{w_2}(p_i)$, in time that mainly depends on the number of edges of E. This is accomplished using the following "ray-shooting" procedure. First we find a starting point z on ∂E by shooting a horizontal ray from $p_i \in E$ and finding the nearest of its intersections with ∂B and ∂R. We next traverse the boundary of E in counterclockwise direction from z as follows. Suppose we have reached some point x on some edge e of ∂B. We shoot a ray from x along e (so that B lies to the left of the ray) and find its intersection, x', with ∂R. If e ends before x', then we turn at the endpoint of e to the adjacent edge, e', along ∂B and repeat shooting along e' towards ∂R. Otherwise, we turn at x' to ∂R in counterclockwise direction, and shoot along the new edge towards ∂B. Repeating this process, we will eventually return to z, thereby completely tracing the boundary of ∂E. (Figure 3.1 illustrates this process.) Since both faces, B and R, are convex each ray shooting query can be carried out in time $O(\log n_v)$ (see [CD]). Thus, the calculation of $F_v(p_i)$ can be

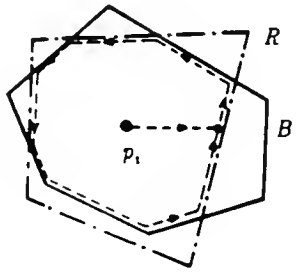


Figure 3.1. Tracing the boundary of the intersection.

accomplished in time $O(|F_v(p_i)|\log n_v)$, $|F_v(p_i)|$ being the number of edges bounding $F_v(p_i)$.

In general, that is, if $M>2$ we apply the merging process $M-1$ times to take into account all M children w_j of v. Since M is a constant depending only on r, the sample size, all faces $F_v(p_i)$ can be obtained in time $O((K(m_v,n_v)+m_v)\log n_v)$.

Finally, we consider the overhead of the top-down construction of the tree $\mathcal{T}$. At each node v we take a random sample of size r of the dual lines that reach v, construct and triangulate their arrangement, split the dual points among these triangles, and determine for each triangle which of the m_v dual lines p_i^* it intersects. Each triangle gives rise to a child of v which is passed the points that fall into the triangle as well as the lines that intersect it. Each step either takes constant (randomized) time or time linear in n_v or m_v.

We can therefore obtain the following recurrence formula for $T(m,n)$, the time needed for m points and n lines. For $m<n^2$ we have

$$T(m,n) \le \sum_{i=1}^{M} T(m_i,n_i) + O((\overline{K}(m,n)+m+n)\log n),$$

where the m_i, the n_i and M satisfy conditions (I) and (II) of the analysis in Section 2, and, with high probability, also condition (III) (namely when the random sample is indeed an ϵ-net). For $m \ge n^2$ we have

$$T(m,n) = O(m\log n),$$

and for $m=0$ we have

$$T(m,n) = O(n\log n).$$

Using the bounds on $\overline{K}(m,n)$ and $K(m,n)$ obtained above, one can obtain the following bound on $T(m,n)$; the proof is a straightforward generalization of the proofs of Lemma 2 and Theorem 3 and is left to the reader.

Theorem 4. The time complexity of the above randomized algorithm for computing m distinct faces in an arrangement of n lines is, with high probability, $T(m,n)=O(m^{2/3-\delta}n^{2/3+2\delta}\log n+n\log n\log m)$, for any $\delta>0$.

Remarks. (1) The algorithm that we obtain is "Las Vegas" randomized since it does not verify that the sample of lines chosen at any node of $\mathcal{T}$ in fact is an ϵ-net. Even if the sample is not an ϵ-net, the algorithm will still run correctly, albeit potentially more slowly. Thus the algorithm will always produce the correct output, and with high probability will have time complexity as stated in Theorem 4. Alternatively, we could verify that the sample set is indeed an ϵ-net thus obtaining a "Monte Carlo" randomized algorithm. This can be done by constructing and triangulating the arrangement of the sample lines and checking whether the number of lines cutting

any one triangle is indeed sufficiently small.[9] This verification step clearly takes only linear time. We can now repeat the random selection until a satisfactory sample is obtained.

(2) The combinatorial considerations in Section 2 can be viewed as analyzing the space complexity of the Monte Carlo version of the algorithm (thus getting a worst-case bound) or, alternatively, the most likely space complexity of the "Las Vegas" version (thus getting a probabilistic bound). This does not mean, however, that the Las Vegas algorithm takes more space in the worst case than in the most likely case – only we did not prove that it does not. In fact, the worst-case space complexity of the Las Vegas version depends on implementation details that we deliberately omit. The issue here is how much of the partition tree we need to store at any point in time. The best result is obtained if we use the partition tree only conceptually and implement the algorithm as a traversal of the tree without ever building it.

(3) The ray shooting technique used in the above algorithm does not seem to generalize to the more complicated task of constructing m faces in an arrangement of n line segments, which is what we study in Sections 4 through 7. The alternative merging technique that we use for line segments, described in Section 7, can also be applied to the simpler case at hand. However, we have chosen to present here the ray shooting technique because of its relative simplicity in the case of convex polygons.

4. The Complexity of Many Faces in an Arrangement of Line Segments

This section extends the analysis given in Section 2 to the case of line segment arrangements, that is, we consider the problem of estimating the maximum combinatorial complexity, $R(m,n)$, of m faces in an arrangement of n line segments in the plane. In contrast to the case of lines where all faces are convex, a face in a line segment arrangement is not necessarily convex and need not even be simply connected (see Figure 4.1). Because of the non-convexity of faces, there is no reason why the maximum number of edges bounding a single face should be at most n. Indeed, the total number of edges bounding a single face can be as large as $\Omega(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann's function, and this bound is tight in the worst case, as was shown in [HS], [PSS], and [WS]. Lines are a special case of line segments, which implies $R(m,n) \geq K(m,n)$. Thus, the lower bound of [EW] for line arrangements extends to line segments, that is, $R(m,n) = \Omega(m^{2/3}n^{2/3})$.

In spite of the technical difficulties caused by the boundedness of line segments, we will obtain an upper bound on $R(m,n)$ that is roughly the same as the bound on $K(m,n)$ obtained in Section 2. Again, the bound will be derived from an analysis of the space complexity of an algorithm for calculating m such faces. In Sections 6 and

[9] A sample of lines is an ϵ-net if *every* (open) triangle that avoids all lines is cut by at most some number of the original lines. The verification step as outlined does not verify that this is true but for our purposes it is enough that every triangle of the triangulation has the desired property.

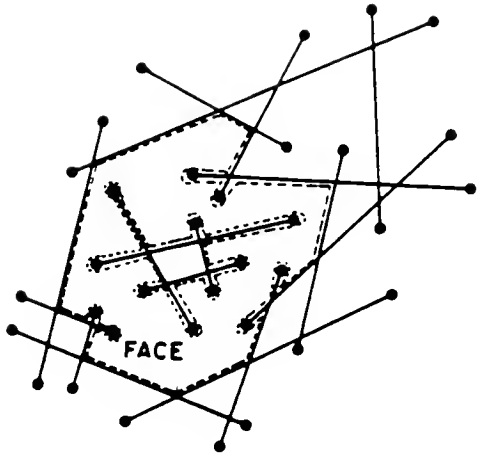


Figure 4.1. A face in a line segment arrangement.

7 we will show how to implement the algorithm so that the calculation of m faces in an arrangement of n line segments takes time that is only slightly more than our upper bound on $R(m,n)$.

Let $S=\{s_1,s_2,\ldots,s_n\}$ be a set of n line segments in the plane, and let $A=A(S)$ be their arrangement. Let $P=\{p_1,p_2,\ldots,p_m\}$ be a set of points that we use to designate faces of A, and consider the problem of calculating the faces of A that contain the p_i. Since we may reach a situation where a face of a subarrangement contains more than one point, we use the auxiliary notation $\bar{R}(m,n)$ to denote the maximum complexity of the faces in an arrangement of n segments that contain m given points (accounting for each face only once). Clearly, $\bar{R}(m,n)=R(m,n)$ whenever both functions are defined.

For each i let l_i denote the line containing s_i, and define $L=\{l_i \mid 1\leq i\leq n\}$. Apply the dual construction given in Section 2 to the lines in L and to the points in P. Thus we recursively build a partition tree $\mathcal{T}$ each of whose nodes v corresponds to an open triangle Δ_v in the dual plane. Node v represents L_v which is the set of lines in L whose dual points lie in Δ_v (and in all triangles corresponding to ancestors of v) and P_v which is the set of points in P whose dual lines intersect Δ_v (and all triangles corresponding to ancestors of v). Implicitly through L_v, v also represents the set of line segments $S_v=\{s_i \mid l_i\in L_v\}$. Consistent with the notation in Section 2, we define $m_v=|P_v|$ and $n_v=|S_v|$. When $m_v\geq\kappa(n_v)$ we stop the construction of $\mathcal{T}$ below node v, pass back to the primal plane, construct there the arrangement $A(S_v)$ of the n_v line segments s_i in S_v, and collect the required faces of $A(S_v)$ that contain the points in P_v. The space complexity of the entire $A(S_v)$, and thus also of the faces in question, is $O(n_v^2)=O(m_v)$.[10] We also stop the construction of $\mathcal{T}$ below v if $m_v=0$. In this

[10]Indeed, the number of vertices, edges, and faces of $A(S_v)$ is proportional to the number of intersecting line segment pairs which is, of course, at most $\binom{n_v}{2}$.

case we construct the unbounded face of the associated line segment arrangement from scratch (see Theorem 11). (Note that an important feature of the construction is that in the dual plane we use the dual points l_i^* of the (unbounded) lines l_i. We thus ignore the fact that we have to consider only the portions s_i of these lines. However, when we pass back to the primal plane, we always process the line segments s_i rather than the lines that contain them.)

We now proceed to the discussion of how the relevant faces of $A(S_v)$ are obtained if $m_v < \kappa(n_v)$. As in the case of lines, we compute $Q_v = P_u - P_v$, u the parent of v. This is the set of points whose dual lines reach u but miss Δ_v. Every point in Q_v either lies above all lines in L_v or below all these lines. In any case, those points lie in the unique unbounded face, F_v^∞, of $A(S_v)$. As mentioned above, this face is bounded by $O(n_v \alpha(n_v))$ edges. (If the given collection S contains also unbounded rays or lines, we may have to consider two unbounded faces of $A(S_v)$, as in the case of lines.)

The main difficulty lies of course in merging (or rather intersecting) the recursively available faces of the children $w_1, w_2, \ldots, w_M$ of v to get the desired faces at v. For a particular point, $p \in P_v$, this means that we construct $F_v(p)$ by intersecting the faces $\tilde{F}_{w_j}(p)$ that contain p in the subarrangements associated with the children of v ($\tilde{F}_{w_j}(p) = F_{w_j}(p)$ if p reaches w_j and $\tilde{F}_{w_j}(p) = F_{w_j}^\infty$, otherwise). Our goal is to obtain an appropriate generalization of the combination lemma for lines (Lemma 1) that will enable us to bound in a similar manner the total complexity of the faces $F_v(p_i)$. This generalization is quite complicated and described in detail in the following section. This section continues with explaining the result of this generalization and using it to complete the combinatorial analysis of many faces in line segment arrangements.

For the remainder of this section (as well as for Sections 5, 6, and 7) we define a *polygon* as an open region in the plane that can occur as a face in a line segment arrangement. Thus, a polygon is neither necessarily convex nor simply connected. The boundary of a polygon consists of one or more connected components called *contour cycles*. We think of a contour cycle as a Jordan curve that may touch but cannot cross itself. In particular, if an edge lies in the interior of the closure of the polygon (it bounds the polygon on both sides), then the contour cycle has two parts that touch along the edge (see Figure 4.1). When we count the number of edges of a polygon we count each such edge portion twice. A vertex of the polygon is *reflex* if the inside angle at this vertex exceeds π.[11] With these definitions we have the following generalization of Lemma 1 – the proof of this lemma, called the "combination lemma for line segments", will be given in Section 5.

---

[11]For our purposes it suffices to assume that the line segments are in general position which, among other things, means that no line segment contains an endpoint of another line segment. A consequence of this assumption is that all reflex angles lie at endpoints of the line segments and the inside angle of every reflex vertex is 2π.

Lemma 5. Let $p_1,p_2,\ldots,p_k$ be points in the plane, and let $\{B_1,B_2,\ldots,B_s\}$ and $\{R_1,R_2,\ldots,R_t\}$ be collections of "blue" and "red" polygons that satisfy the following three conditions.

(i) The blue (red) polygons are pairwise disjoint, the total number of blue (red) edges is β (ρ), and the total number of reflex vertices is r.

(ii) Each point p_i is contained in a blue polygon B_{s_i} and a red polygon R_{t_i}.

(iii) If for each $1\le i\le k$ we define E_i to be the connected component of $B_{s_i}\cap R_{t_i}$ that contains p_i (see Figure 5.3), then $E_i\neq E_j$ if $i\neq j$.

Then the total number of edges of the E_i is at most $\beta+\rho+O(k)+O(r)$.

Using Lemma 5, we can complete the analysis of the merge step at some node v of tree T. Applying Lemma 5 a constant number of times we conclude that the overall complexity of the faces $F_v(p_i)$ is at most the sum of the complexities of all the faces $\tilde{F}_{w_j}(p_i)$, $i=1,2,\ldots,m_v$ and $j=1,2,\ldots,M$, plus $O(m_v)+O(r_v)$, where r_v is the number of reflex vertices in all faces $\tilde{F}_{w_j}(p_i)$. But each such reflex vertex must be an endpoint of one of the n_v segments in S_v, so $O(r_v)=O(n_v)$. In addition to the complexity of the recursively available faces $F_{w_j}(p_i)$ we need to take into account the total number of edges bounding the unbounded faces $F_{w_j}^{\infty}$, which is at most

$$\sum_{j=1}^{M} O(n_{w_j}\alpha(n_{w_j})) = O(n_v\alpha(n_v)) .$$

We thus obtain the following recurrence formula for $\overline{R}(m,n)$.

$$\overline{R}(m,n)\le \begin{cases} 0 & \text{if } m=0 \\ am & \text{if } m\ge\kappa(n) \\ \sum_{i=1}^{M}\overline{R}(m_i,n_i)+bm+b'n\alpha(n) & \text{if } m<\kappa(n) \end{cases}$$

for some constants $a,b,b'>0$, where the m_i, n_i, and M satisfy the conditions (I) through (III) stated in Section 2. A proof that is almost identical to the proof of Lemma 2 (which is therefore omitted) implies the following solution of the recurrence relation.

Lemma 6. $\overline{R}(m,n)\le Dm^{2/3-\delta}n^{2/3+2\delta}+Am+Bn\alpha(n)\log m$, for any $\delta>0$, where the coefficients A,B,D depend on δ.

Remark. A major feature of Lemma 5 (and of its simpler variant Lemma 1) is that the terms β and ρ appear with multiplicative constant 1 in the bound for the total combinatorial complexity of the E_i. This ensures that the recurrence formula for

$\overline{R}(m,n)$ given above involves the term $\sum_{i=1}^{M} \overline{R}(m_i,n_i)$ with multiplicative constant 1. This is essential for obtaining the bound stated in Lemma 6.

If $m \leq \kappa(n)$ (that is, at most one point per face in the original arrangement is specified), then

$$m \leq m^{2/3-\delta} n^{2/3+2\delta},$$

so that we can drop the second term from the bound in Lemma 6. We thus conclude with the main result of this section.

Theorem 7. The maximum number of edges of m faces in an arrangement of n line segments is

$$R(m,n) = O(m^{2/3-\delta} n^{2/3+2\delta} + n\alpha(n)\log m),$$

for any $\delta > 0$.

Remark. It is easy to check that $\overline{R}(m,n) = O(m^{2-\delta} + n\alpha(n)\log m)$, for any $\delta > 0$, also satisfies the recurrence relation derived for $\overline{R}$. This constitutes a weak generalization of Canham's theorem [Ca] for lines to the case of line segments.

5. Proof of the Combination Lemma for Line Segments

In this section we provide a proof of Lemma 5, the combination lemma for line segments. This is the crucial lemma in the analysis of the complexity of many faces in a line segment arrangement presented in Section 4. We proceed by considering the interaction between a blue and a red polygon, a blue polygon and many red polygons, and finally many blue and many red polygons. The results in this section have a topological and combinatorial flavor and add up to a proof of Lemma 5.

The main concept in this section is that of a *polygon* which is defined general enough so that every face in a line segment arrangement passes as a polygon. As mentioned in Section 4, a polygon is thus connected but not necessarily simply connected, and its boundary consists of connected components which we call *contour cycles*. We can avoid the technical difficulty caused by the fact that a connected component of the boundary need not be a simple Jordan curve,[12] if we replace each line segment by a rectangle of sufficiently small width. For small enough widths we

[12] A *(simple) Jordan curve* has the property that every sufficiently small disk whose center lies on the curve is cut into two connected components if we remove from it all points of the curve. This implies that a Jordan curve is either unbounded at both ends or it is bounded in which case it is said to be *closed*. A connected piece of a Jordan curve is called a *Jordan arc*; it satisfies the same condition as the Jordan curve except at its two endpoints at which removing the points of the arc leaves every small enough disk connected.

get the same intersection pattern for the rectangles as for the line segments, and a face (a connected component of the plane minus the union of all rectangles) is now bounded by contour cycles that are simple Jordan curves. For technical reasons we direct each contour cycle so that the polygon it bounds lies to its left. Thus, a contour cycle that delimits a hole of the polygon is directed in clockwise order whereas the outside contour cycle (if it exists) is directed in counterclockwise order.

Our first result is topological and asserts that the traversal of every contour cycle of a connected component E of $B \cap R$, B a "blue" and R a "red" polygon, "agrees" with the traversal of the contour cycles of B and R. By this we mean that the common points of a contour cycle of E and one of B (or R) are traversed in the same order independent of whether we follow the contour cycle of E or that of B (or R).

Lemma 8. Let E, B, and R be polygons such that E is a connected component of $B \cap R$, and let a,b,c be three points on $\gamma \cap \xi$, where γ is a contour cycle of E and ξ is a contour cycle of B. The order of points a,b,c along ξ is the same as along γ.

Proof. Note first that the directions of γ and ξ along common boundary pieces agree since E and B lie on the same side of these pieces. Take ξ, the contour cycle of B that contains points a,b,c, and let ab_ξ, bc_ξ, and ca_ξ be the pieces (Jordan arcs) of ξ from a to b, from b to c, and from c to a. By assumption, points a,b,c belong also to γ. If ab_ξ is contained in γ then the assertion is trivially true since the traversal from a to b on γ only passes points of ab_ξ, and c does not belong to ab_ξ. Otherwise, ab_γ, the portion of γ leading from a to b, contains pieces that do not belong to ξ – these pieces are necessarily contained in the union of ∂R and $\partial B - \xi$ (see Figure 5.1). Let δ be such a piece, that is, δ is a maximal connected component of $\gamma - \xi$, whose starting point, z, lies on ab_ξ. We prove below that the endpoint, w, of δ lies also on ab_ξ which implies the lemma since we cannot reach any point of $\xi - ab_\xi$ before passing through b.

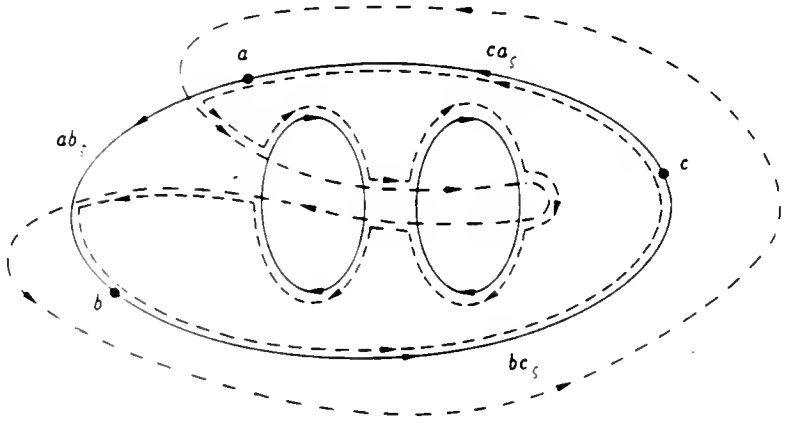


Figure 5.1. Traversing the boundary of the intersection.

Assume that w does not lie on ab_ξ. Let δ' be the portion of ξ leading from z to w, including the endpoints (see Figure 5.2; note that δ' must contain b). By construction, δ and δ' are disjoint and their union is a closed Jordan curve σ. Let a' and b' be two interior points of E that lie sufficiently close to a and b. It is thus possible to connect a' and b' by a Jordan arc, α, that lies sufficiently close to ab_ξ such that $\alpha \cap \delta' = \emptyset$ and α and δ cross in a single point. Thus, a' and b' lie in different components of $R^2 - \sigma$ which is impossible since σ is disjoint from E and E is connected (see Figure 5.2). □

Remark. Lemma 8 expresses a consistency property of intersections of polygons. Among other things it implies that if an edge e of B (or R, for that matter) contains several edges of E then these edges appear in the same order along e and along the relevant contour cycle of E.

Consider next a blue polygon B with ℓ contour cycles $\xi_1, \xi_2, \ldots, \xi_\ell$ and let $p_1, p_2, \ldots, p_m$ be the points designating m desired regions contained in B. We let R_i be the red polygon that contains p_i, for $1 \leq i \leq m$, and we write E_i for the connected component of $B \cap R_i$ that contains p_i (see Figure 5.3). To be consistent with the assumptions for Lemma 5 we allow $R_i = R_j$ but we assume that $E_i \neq E_j$ if $i \neq j$. We will analyze the blue boundary pieces of the E_i by constructing a graph, $\mathcal{G}_B$, and proving certain properties about $\mathcal{G}_B$. This graph will be instrumental in proving an upper bound on the total number of blue edges bounding the E_i.

We need a few definitions. A *blue boundary piece* of E_i is a connected component of $\partial E_i \cap \partial B$. Every blue boundary piece, δ, belongs to a contour cycle γ of some E_i and to a blue contour cycle ξ_j. Since contour cycles are directed, we can define a *predecessor* and a *successor* of δ in both cycles, which are the blue boundary pieces immediately before and after δ in γ and in ξ_j. Note that it is possible that the predecessor (successor) of δ in γ is the same as in ξ_j, but this is not necessarily the case. $\mathcal{G}_B$ is a graph whose nodes are the points $p_1, p_2, \ldots, p_m$ and additional ℓ points

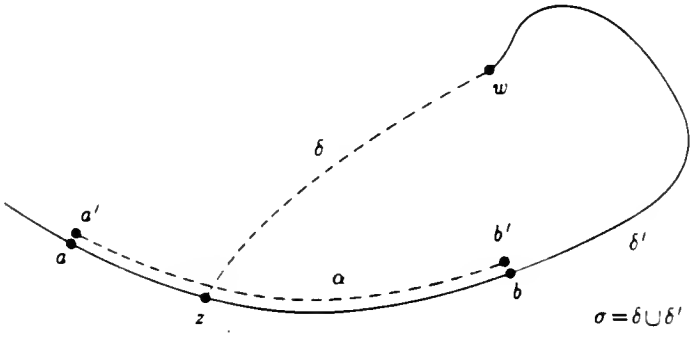


Figure 5.2. An impossible configuration.

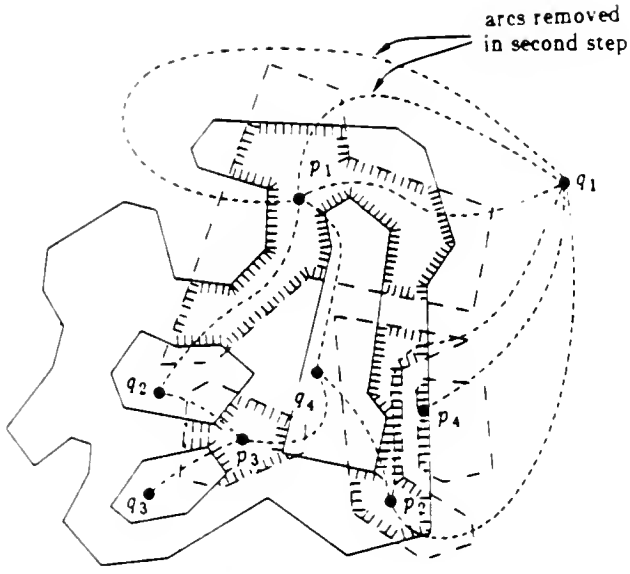


Figure 5.3. The graph $\mathcal{G}_B$.

$q_1, q_2, \ldots, q_\ell$ so that q_i lies in the interior of the connected component of $R^2 - B$ bounded by ξ_i; we denote this component by X_i. For every blue boundary piece $\delta \subset \partial E_i \cap \xi_j$, $1 \le i \le m$ and $1 \le j \le \ell$, we draw a curve connecting p_i with q_j. This curve, a plane embedding of an arc of $\mathcal{G}_B$, connects an arbitrary point on δ with p_i and q_j. Since δ is part of the common boundary of E_i and X_j we can draw the curve completely within $E_i \cup X_j$. The connectedness of each E_i and X_j implies, by a consequence of Schönfließ' theorem [Mo], that we can draw all such edges without any crossings. Hence, $\mathcal{G}_B$ is planar.

The second step of the construction removes sufficiently many of the duplicate arcs (connecting the same two points) so that we can apply Euler's relation to derive an upper bound on the number of remaining arcs. Whenever two arcs of $\mathcal{G}_B$ connect the same two points and they correspond to two blue boundary pieces such that one is the successor of the other in both contour cycles, then we delete the successor arc. (In the case that both blue boundary pieces are successors of each other, we make an arbitrary decision to break the tie.) As a result of this deletion operation, every blue boundary piece $\delta \subset E_i \cap \xi_j$ intersects an arc of $\mathcal{G}_B$, unless there is another such piece $\delta' \subset E_i \cap \xi_j$ that precedes δ in both contour cycles. Note that the removal of arcs as described does not eliminate all multiarcs but it guarantees that, barring one extreme situation mentioned below, every region of the embedding of $\mathcal{G}_B$ is bounded by at least four arcs, counting an arc twice if it lies in the closure of the region. The one case where this does not hold is when $\mathcal{G}_B$ contains only two vertices and one connecting (doubly counted) arc. This arises when B contains just one point p_i, and the corresponding E_i uses only one contour cycle of B. In this case we can delete this arc too, because it will not be used in the argument to follow. The factor "four" rather than "three" which is typical for planar graph arguments can be used because $\mathcal{G}_B$ is bipartite by definition and has therefore no odd cycles. Using Euler's relation we

derive that the number of remaining arcs is at most $2(m+\ell)-4$ (also in the special case mentioned above).

We are now ready to prove a strong variant of Lemma 5 stating that the k connected components of the $B_i \cap R_j$, $1 \leq i \leq s$ and $1 \leq j \leq t$, containing the k points $p_1, p_2, \ldots, p_k$ are bounded by a total of at most $\beta+\rho+r+8k+4\ell-16$ edges, where

β is the total number of (blue) edges of the B_i,
ρ is the total number of (red) edges of the R_j,
r is the total number of reflex vertices of the B_i and R_j, and
ℓ is the total number of contour cycles of the B_i and R_j.

Lemma 5 is implied because $\ell \leq r+2k$ (each B_i or R_j has at most one exterior contour cycle, there are at most $2k$ blue and red polygons, and each interior contour cycle must contain a reflex vertex), and thus $\beta+\rho+r+8k+4\ell-16 = \beta+\rho+O(r)+O(k)$. In the argument to come we traverse all blue (and symmetrically all red) contour cycles and count the blue (red) edges of the E_i, $1 \leq i \leq k$, as we encounter them, by charging them to various "accounts". Note that a blue (red) edge can contain several such edges. We define $\mathcal{G}$ as the union of all graphs $\mathcal{G}_{B_i}$ and $\mathcal{H}$ as the union of all graphs $\mathcal{G}_{R_j}$ defined symmetrically for all red polygons.

The easy case is if a blue (red) edge, e, contains at most one edge of all the E_i – the appearance of this edge is accounted for by the term β (ρ) in the upper bound. Otherwise, let e_1 and e_2 be two consecutive components of $e \cap (\bigcup_{1 \leq i \leq k} \partial E_i)$, and assume that e_1 is already accounted for. We assume e is blue.

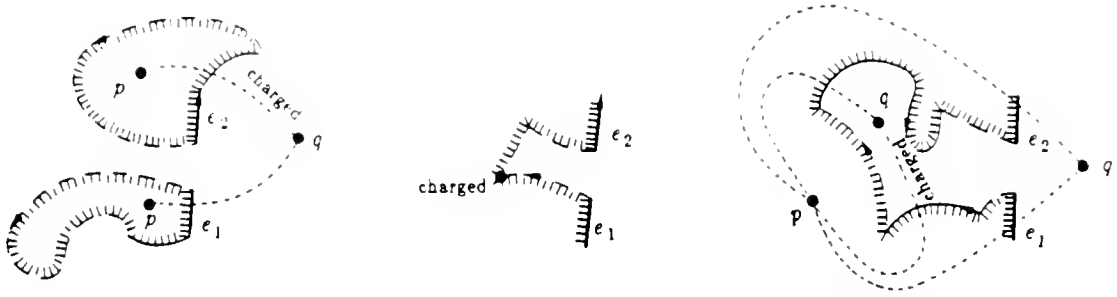
(i) If e_1 and e_2 do not lie on a common contour cycle of an E_i (and thus belong to the boundaries of two different polygons E), then we charge e_2 to the arc of $\mathcal{G}$ that is induced by the blue boundary piece that contains e_2 (note that this arc cannot have been deleted from $\mathcal{G}$; see Figure 5.4(a)).

On the other hand, if e_1 and e_2 belong to a common contour cycle γ of some E_i, then we distinguish two cases. Let γ_0 be the piece of γ connecting the last point of e_1 with the first point of e_2.

(ii) If γ_0 contains no blue boundary piece, then we charge e_2 to the (first) reflex red vertex that lies on γ_0 – such a vertex must exist since, otherwise, we could not be led back to the same blue edge (see Figure 5.4(b)).

(iii) Otherwise, we charge e_2 to the arc of $\mathcal{G}$ that is induced by the first blue boundary piece on γ_0 (see Figure 5.4(c)).

It is easy to see that each reflex vertex is charged at most once. The definition of $\mathcal{G}$ guarantees that in each case where we charge $\mathcal{G}$ there is in fact an arc that takes the charge. We now argue that the mechanism we use can charge an arc of $\mathcal{G}$ at most twice. In case (i) the arc takes the charge for an edge, e_2, that is the first edge of the corresponding blue boundary piece. Since every blue boundary piece has only one first edge, this case can occur only once. In case (iii), the arc that takes the charge for



(a) Charging rule (i). (b) Charging rule (ii). (c) Charging rule (iii).

Figure 5.4.

a pair of edges, e_1 and e_2, corresponds to a different blue contour cycle. However, e_1, the blue boundary piece δ that corresponds to the charged arc, and e_2 all belong to the same contour cycle of some E_i, and δ is the first blue boundary piece that occurs between e_1 and e_2 in this cycle. Clearly, it cannot be the first blue boundary piece after a blue boundary piece other than that containing e_1, which implies that also this case puts at most one charge onto any one arc. Thus, summing over all blue and red edge duplications, we see that the total number of edges bounding the E_i, $1 \leq i \leq k$, is at most

$\beta+\rho+r$ plus twice the total number of arcs of $\mathcal{G}$ and $\mathcal{H}$.

The number of arcs of $\mathcal{G}$ and $\mathcal{H}$ is at most $4k+2\ell-4s-4t$ since every point p_i is counted twice (once for the blue and once for the red polygon that contains it). This implies the claim and completes the proof of Lemma 5.

6. Calculating Many Faces in an Arrangement of Line Segments

We next turn to the task of calculating the faces in an arrangement of n line segments, $s_1,s_2,\ldots,s_n$, that contain m given points, $p_1,p_2,\ldots,p_m$. Generalizing the ray shooting method used in Section 3 for line arrangements is problematic because it would call for performing such ray shooting queries inside polygonal regions that are not simply connected. No efficient technique for doing so is currently known. This section presents an alternative approach based on the line sweeping technique (see e.g. [PS]).

Consider the algorithmic issues that arise in efficiently implementing the algorithm implicitly described in Sections 4 and 5. For any node v that satisfies $m_v \geq \kappa(n_v)$ (thus, v corresponds to a call at the bottom of the recursion) we need to calculate the entire arrangement of the n_v line segments in S_v, and extract from it the faces containing the m_v points reaching v. Using the sweep algorithm in BO this arrangement can be constructed in $O(n_v^2 \log n_v)$ time, and the faces that contain the m_v points can be identified in time $O(m_v \log n_v)$ using any of a number of efficient

point location methods. Hence, such a node v can be processed in time $O(m_v \log n_v)$. If $m=0$ (also this case corresponds to a call at the bottom of the recursion), the unbounded face can be constructed in time $O(n\log^2 n)$ (see Theorem 11).

For an inner node v (that is, v corresponds to an intermediate recursive call) we need to calculate the exterior face of $A(S_v)$ which contains the points in Q_v by definition. This step of the computation will be described at the end of Section 7. It will fall out as a special case of the general merge procedure described there.

As for the general merging step at v, let $p_1,p_2,\ldots,p_{m_v}$ be the points that reach v. For each p_i we need to calculate the face $F_v(p_i)$ of the arrangement $A(S_v)$ that contains p_i. This face is the connected component containing p_i of the intersection of the M faces $\tilde{F}_{w_j}(p_i)$, $j=1,2,\ldots,M$, where $\tilde{F}_{w_j}(p_i)=F_{w_j}(p_i)$ if p_i reaches the child w_j of v, and $\tilde{F}_{w_j}(p_i)=F_{w_j}^{\infty}$ if $p_i\in Q_{w_j}$. As in Section 3, our goal is to construct these faces in time that only depends on their total combinatorial complexity. Section 7 will present a method, called the *blue–red merge*, that can be used to construct the $F_v(p_i)$ within the desired time bound.

Let us be more specific about the blue–red merge. The input to this procedure is a set of pairwise disjoint blue (and red) polygons bounded by a total number of $\beta(\rho)$ edges, and a set of k points each one inside some blue and some red polygon. The output is the set of k polygons that are the connected components of the blue–red intersections that contain the k points. The blue–red merge will construct the output polygons in time $O((\beta+\rho+k)\log(\beta+\rho+k))$ (see Theorem 10). To construct the $F_v(p_i)$ we apply the blue–red merge $M-1$ times to the $\tilde{F}_{w_j}(p_i)$, $j=1,2,\ldots,M$. Using the blue–red merge we will also be able to construct the unbounded face of a set of n line segments in time $O(n\alpha(n)\log^2 n)$, and in time $O(n\alpha(n)\log n)$ if we assume that it can be constructed as a connected component of the intersection of a constant number of precomputed unbounded faces (see Theorem 11 and the second remark following it).

The recursive processing of a node v in $\mathcal{T}$ thus consists of calculating the $\tilde{F}_v(p_i)$ from the recursively computed faces of the subarrangements $A(S_{w_j})$ as well as constructing the unbounded face, F_v^{∞}, of v. The time required to construct the unbounded face is $O(n_v\alpha(n_v)\log n_v)$ (see also Section 7). The amount of time required at node v is thus

$$\sum_{j=1}^{M} O((\overline{R}(m_{w_j},n_{w_j})+m_{w_j})\log n_{w_j}) + O(n_v\alpha(n_v)\log n_v) =$$

$$= O((\overline{R}(m_v,n_v)+m_v+n_v\alpha(n_v))\log n_v).$$

Here we use the fact that the number of reflex vertices in the relevant faces is $O(n_v)$ – they must be endpoints of line segments in S_v. In addition, we use Lemma 5 for each of the $M-1$ blue–red merging steps to deduce that the output size of each merge is linear in its input size. From this the above bound follows readily. We now put everything together, taking also into account the probabilistic overhead of constructing $\mathcal{T}$ (as in Section 3).

Let $T(m,n)$ denote the time needed by our algorithm to calculate the faces in an arrangement of n line segments that contain m given points. If we assume that the random samples that we draw are indeed ϵ-nets (which is true with high probability) we get the following recurrence relation for $T(m,n)$.

$$T(m,n) = O(n\log^2 n) \text{ if } m = 0,$$

$$T(m,n) = O(m\log n) \text{ if } m \geq \kappa(n), \text{ and}$$

$$T(m,n) \leq \sum_{i=1}^{M} T(m_i,n_i) + O((R(m,n)+m+n\alpha(n))\log n) \text{ if } m < \kappa(n),$$

where the m_i, the n_i, and M satisfy the conditions (I) through (III) stated in the analysis given in Section 2. Using the bounds on $R(m,n)$ and $R(m,n)$ obtained above, one can easily obtain the following bound, in much the same way as in the proofs of Lemma 2 and Theorem 3.

Theorem 9. The m faces designated by m points in an arrangement of n line segments in the plane can be constructed in probabilistic time

$$T(m,n) = O(m^{2/3-\delta}n^{2/3+2\delta}\log n + n\alpha(n)\log^2 n\log m),$$

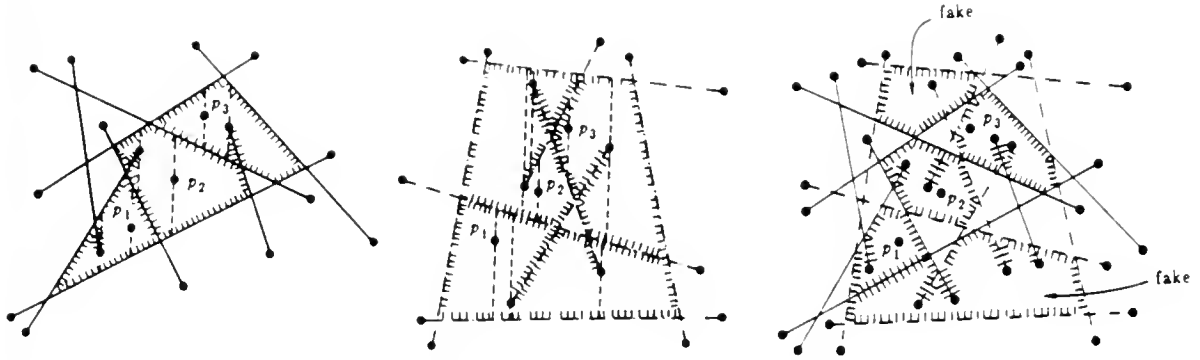
where δ is any positive real number.

Remark. See the discussion on the probabilistic nature of our result at the end of Section 3. The same remarks apply to the algorithm presented in this section.

7. The Blue–Red Merge

This section presents the details of the blue–red merge which computes the relevant faces at a node v of T assuming that the faces at the M children of v have already been constructed recursively. As in Section 3 we assume $M = 2$, so that we need to intersect only two faces around each point $p_i \in P$, $1 \leq i \leq k$. We are thus given two collections of (not necessarily simply connected) open polygons in the plane, $\{B_1,B_2,\ldots,B_s\}$ and $\{R_1,R_2,\ldots,R_t\}$; the B_i are called the *blue* polygons and the R_j are the *red* polygons. We can assume that any two blue (red) polygons are disjoint. See Figure 7.1(a,b) for an illustration of this set-up. Let β and ρ denote the total number of edges of the blue and red polygons, respectively.

Our goal is to calculate all polygons E_i, where, for each $p_i \in P$, E_i is the connected component containing p_i of the intersection of the red polygon and the blue polygon that contain p_i. The resulting E_i are called the *purple* polygons, as each is covered by a red and a blue polygon. Figure 7.1(c) shows the purple polygons that arise from the blue and red polygons as well as the points shown in Figure 7.1(a,b). As in Section 3, we do not exclude the possibility that $E_i = E_j$ for $i \neq j$. If π is the total number of purple edges, then $\pi = \beta+\rho+O(k)+O(r)$, by Lemma 5, where r is the



(a) The blue polygons. (b) The red polygons. (c) The purple polygons.

Figure 7.1. Intersecting blue and red polygons.

total number of reflex blue and red vertices. Since $r \leq \beta+\rho$ we have $\pi = O(\beta+\rho+k)$.

To facilitate the merge, we require certain information to be precomputed and available for each collection of polygons. Specifically, let P_B and P_R be the set of reflex vertices of the blue and red polygons, respectively. We require that each blue polygon B_i be subdivided into convex regions by drawing vertical rays from each point $p \in (P \cup P_B) \cap B_i$ and stopping them as they encounter an edge of B_i; we call the resulting vertical line segment through p the *blue vertical divider* of p, and denote it by $blue(p)$ (see Figure 7.1(a)). Symmetrically, we require that each red polygon is decomposed into convex regions by *red vertical dividers* $red(p)$, $p \in P \cup P_R$ (see Figure 7.1(b)). These convex decompositions of the blue and red polygons will be available recursively.

A particular convex blue (or red) region terminates on the left or the right either

(i) because of a point of P,
(ii) because of a blue (or red) reflex vertex, or
(iii) because of a locally x–extremal (non–reflex) vertex of the corresponding polygon.

The blue–red merge will produce a similar decomposition of the purple polygons into convex regions, which we call the *purple regions.*

We construct the purple regions by sweeping a vertical line across the plane. A purple region starts (and ends) at

(i) a point of P,
(ii) a blue or red reflex vertex,
(iii) an x–extremal vertex of a blue or red polygon, or
(iv) the intersection of a blue and a red edge.

In a left–to–right sweep we will discover the portion of each purple polygon that is to the right of the leftmost point in $P \cup P_B \cup P_R$ that lies in it. Afterwards, in a right–to–left sweep, we will get the portion of each purple polygon to the left of the rightmost point in $P \cup P_B \cup P_R$ that lies in the polygon. Together, the two sweeps discover all edges of the purple polygons. In fact, the sweeps will construct slightly more than the purple regions (the convex decomposition of the purple polygons) and we will have to remove superfluous regions after the two sweeps (see Figure 7.1(c)). More about this later.

We are now ready to describe the left–to–right sweep – the right–to–left sweep is symmetric. We start by constructing a priority queue that stores $P \cup P_B \cup P_R$ ordered by x–coordinates. This priority queue will be referred to as the *event schedule*. During the sweep we maintain separate data structures for the blue, red, and purple regions. Thus, it is convenient to think of a blue, red, and purple plane swept simultaneously and independently. The main purpose of keeping the blue and red regions separate is to avoid processing "uninteresting" blue–red intersections that do not contribute to the boundary of a purple region. The only data structure that represents blue, red, and purple data mixed together is the event schedule. The main part of the blue–red merge is to detect intersections of blue and red edges that occur on the boundary of a purple region. When such an intersection is detected it is added to the event schedule and appropriate actions are taken to adjust the data structures supporting the sweep.

Introduction of scouts. Each time a point $p \in P \cup P_B \cup P_R$ is encountered, we possibly start one or two new purple regions in the purple plane. If $p \in P$ then it belongs to a blue and a red region and we start one new purple region. If p is a reflex vertex of a blue (red) polygon we check whether it lies inside a red (blue) polygon. If it does not we simply discard it, and if it does we start two or one purple region(s) depending on whether or not both incident edges of p lie to the right of the vertical line through p.[13] Whenever a new purple region is started we create two *scouts*, an *upper* and a *lower* scout for the region. The job of the two scouts is to walk along the upper and lower boundary of the new purple region and to look out for events to come which might influence the shape of their region. The scouts always stay with the sweep line and never stroll ahead or stay behind. We describe the way the upper scout of the purple region does its job – the lower scout behaves symmetrically.

The upper scout, u, starts on the lowest blue or red edge above p, the point that gives rise to the purple region at hand. If $p \in P$ then the edge is either the blue edge that contains the upper endpoint of $\mathit{blue}(p)$ or the red edge that contains the upper endpoint of $\mathit{red}(p)$. If p is a blue (red) reflex vertex, then we consult the data

[13]Assuming that we deal with polygons bounded by simple Jordan curves there are exactly two such edges. In our application, however, each reflex vertex is an endpoint of a line segment and thus incident to only one edge. We treat the two sides of this edge as two edges. Thus, either "both incident edges" are to the left or the right of the vertical line through the reflex vertex.

structure that represents the red (blue) sweep to find the lowest red edge vertically above p[14] and we compare this edge with the blue (red) edge that contains the upper endpoint of $blue(p)$ ($red(p)$). The scout moves right along that edge following the sweep line and it watches out for certain events that might influence the construction of the purple boundary.

Without loss of generality, assume that u currently lies on a blue edge. Most importantly, u looks up to the lowest red edge above – call it e_r. Since u is a point of a purple region, all points between u and e_r including u lie in the red polygon bounded by e_r. The reason for u's concern is that, at some future point in time, the red boundary above u might drop below the blue boundary u is currently following. If this indeed happens u will switch to the red boundary which then will delimit the purple region. However, there might be another scout, v, already watching the same red edge from below. In that case only the higher of u and v needs to watch e_r – the other scout can rest, since it is protected by the other scout, who will have to warn it in case the red boundary drops unexpectedly.

In more technical detail, "watching e_r" means that u determines whether e_r and e_b, the edge it walks on, intersect to the right of the sweep line. If so it adds the intersection point to the event schedule of the sweep. If such an intersection does not exist and u reaches the right endpoint of e_b (or e_r) it continues walking on (watching) the next blue (red) edge, if it exists. On the other hand, if e_b and e_r intersect, then u switches over to e_r when the sweep line passes the intersection. In this case, u starts watching e_b which is now the lowest blue edge above u. Of course, there is also the case that the red boundary watched by u discontinuously changes at some point – in this case u must look for a new assignment. Below, this case is treated in detail.

Another job of u is to watch its partner, the lower scout of the same purple region. This is because when the two scouts meet then the purple region ends. Of course, the region ends earlier if another point of P or a reflex vertex is encountered between the two scouts.

Scout invariants. There are two key properties that are satisfied at any given time. The first is that at any point in time each blue and each red edge is watched by at most one upper or lower scout. Of course, the assignment of the scouts may change and, over time, a single edge can be watched by many scouts. The second property is that a blue edge is watched only by scouts that walk along red edges, and a red edge is watched only by scouts walking along blue edges.

Changing assignments. Reassignments for scouts are necessary when new purple regions start and old ones end. A purple region might end, for example, when its two scouts meet. This occurs, for instance, when the rightmost vertex of a blue region lies

[14]This data structure is a balanced tree that stores the red (blue) edges that currently intersect the sweep line. Thus, logarithmic time (in the number of red (blue) edges) is sufficient to find the lowest red (blue) edge above a given point. New edges can be added and old edges can be removed in logarithmic time.

inside a red region. In this event, the two purple scouts are dismissed. However, some transfer of watching responsibility is indicated before the two scouts leave the scene. If the dismissed upper scout, u, was watching a red edge, e_r, then we must check v, the next upper scout below u. If v is idle (this can only be because u protected v from e_r), then v takes over the responsibility to watch e_r. If v is already watching another red edge, then we leave it undisturbed as its red edge must lie below e_r.

The two scouts of a purple region also come together when a purple region ends at the intersection of a red and a blue edge. Any reassignment of watching responsibilities are handled as before.

If a purple region ends because of a point $p_j \in P \cup P_B \cup P_R$ that appears between the scouts, then again the two scouts are dismissed. In this case, however, they will generally be replaced by new scouts employed to guard the purple region(s) started at the vertical divider of p_j.

The reassignment of watching responsibility necessary when we sweep through such a point p_j is done as follows. At the time p_j is encountered, one or two new purple regions are created. Suppose for simplicity that there is only one new region, let u be its upper scout and assume that u starts out on a blue edge. First, u finds the lowest red edge above it using the data structure that represents the sweep in the red plane. Second, u consults the upper scout, w, immediately above and the upper scout, v immediately below. If w is watching the same red edge as u then u forgets about its assignment and becomes idle. Otherwise, u takes up its assignment and checks whether v watches the same red edge. If yes, then v becomes idle. Similar actions are taken when two new purple regions are spawned at p.

Finally, some transfer of watching responsibility may be required at a blue–red crossing lying, say, on the upper boundary of some purple region. Suppose the corresponding upper scout, u, walks along a blue edge, e_b, just before the intersection occurs. To the right of the crossing u follows a red edge, e_r. As noted above, u now starts watching e_b, but we also need to check the upper scout v immediately below u, who might want to start watching e_r. Details are similar as in the cases considered above.

Analyzing the blue–red sweep. The scouts simply trace the boundaries of the purple regions. Each scout adds the intersections between blue and red edges that it predicts to the event schedule. New events are added only when we sweep through a point in P, through a blue vertex, through a red vertex, or through the intersection between a blue and a red edge that is also a vertex of a purple region. Moreover, at each such point only a constant number of additions of new events are made. Thus the total number of events ever scheduled is proportional to the total input and output size, which, by Lemma 5, is $O(\beta+\rho+k)$. The time to add or remove an entry to or from the event schedule is logarithmic in its size which is thus $O(\log(\beta+\rho+k))$. The additional operations involve updating the balanced trees that represent the blue, red, and purple cross-sections along the sweep line, creating and dismissing scouts, and reassigning watching responsibilities. It is plain that we need to perform only

$O(\beta+\rho+k)$ such operations, and that each one can be carried out in time $O(\log(\beta+\rho+k))$.

There is a minor point to be clarified here: the algorithm constructs more than only the "true" purple regions (those that are connected by sequences of adjacencies across vertical dividers to regions that contain points of P). Indeed it constructs all connected components of the blue-red intersections that contain a point of P or a reflex blue or red vertex (see for example Figure 7.1(c) which shows two purple faces that contain no point of P). The construction of these additional faces seems necessary since a genuine purple face can go back and forward in a serpentine-like fashion, which makes it difficult for a sweep to capture its entire boundary unless it collects various portions of the face in advance. Since the number of reflex vertices is at most $\beta+\rho$, this does not affect the asymptotic time-complexity of the above algorithm. The final step now removes fake purple regions. This can be done by a simple graph search in time proportional to the number of purple regions produced by the algorithm, hence in time $O(\beta+\rho+k)$. Thus the blue-red merge takes time $O((\beta+\rho+k)\log(\beta+\rho+k))$.

Theorem 10. Let the B_i (R_j) form a collection of pairwise disjoint blue (red) polygons in the plane bounded by a total number of β (ρ) edges, and let P be a set of k points each contained in a blue and a red polygon. The connected components of the intersections between the blue and the red polygons that contain the given points can be constructed in time $O((\beta+\rho+k)\log(\beta+\rho+k))$.

We next show how to apply the blue-red merge to the calculation of a single face, F, in an arrangement of a collection of n line segments, $S=\{s_1,s_2,\ldots,s_n\}$. As usual, F is assumed to be represented by a single point p contained in F. We now employ a straightforward divide-and-conquer procedure (nothing like our present intricate partition tree scheme).

Step 1. Partition S into subsets S_1 and S_2 of about half the size of S each.
Step 2. Calculate the faces F_1 and F_2 in the arrangements $A(S_1)$ and $A(S_2)$ that contain p.
Step 3. Apply the blue-red merge described above to F_1, F_2 and $\{p\}$.

By the results of [PSS], we know that F_1 and F_2 together have $O(n\alpha(n))$ edges. Thus, by Theorem 10, Step 3 constructs F from F_1 and F_2 in time $O(n\alpha(n)\log n)$. We therefore get

$$T(n) = 2T(n/2)+O(n\alpha(n)\log n) = O(n\alpha(n)\log^2 n)$$

for the total amount of time required to construct F. We state this result as a theorem.

Theorem 11. Any single face in an arrangement of n line segments in the plane

can be calculated in time $O(n\alpha(n)\log^2 n)$.

Remarks. (1) Theorem 11 extends and simplifies previous results on constructing a single face in a line segment arrangement obtained in [PSS].

(2) In the construction of m faces in a line segment arrangement, we need Theorem 11 for calculating the unbounded face, F_v^∞, in $A(S_v)$, for every node v of T. If v is an inner node, then we already have a constant number of unbounded faces available such that F_v^∞ is the unbounded connected component of their intersection. Hence, we do not have to pay for the recursive overhead (as in Theorem 11) which gives us an $O(n\alpha(n)\log n)$ time algorithm for constructing F_v^∞. No $\log n$ factor can be saved if v is a leaf of the tree.

8. Discussion and Open Problems

In this paper we have obtained almost tight upper bounds for the maximum number of edges bounding m faces in an arrangement of n lines or line segments. We also presented efficient randomized algorithms for the calculation of these faces. The randomized time-complexity of these algorithms is only slightly higher than the upper bounds on the number of edges that are to be reported. The main technical tools that we have introduced and used in our analysis are

(i) the construction of a customized partition tree for a set of n points and m query lines, using a random sampling technique similar to those of [HW] and [Cl], and

(ii) the combination lemmas for faces in arrangements of lines and of line segments.

This final section concludes with some comments on our techniques and states related open problems.

The algorithm presented in Sections 2 and 3 and its analysis extend simpler techniques, also based on range searching partitioning schemes, which have been recently employed to solve other problems on the interaction between points and lines in the plane. One such problem, originally posed by Hopcroft, is the following.

Given m points and n lines in the plane, determine whether any of the points lies on any of the lines.

This problem was solved by Cole, Sharir and Yap [CSY] in time approximately on the order of $m^{\frac{2\alpha}{\alpha+1}} n^{\frac{1}{\alpha+1}}$, where $\alpha = \log_2 \frac{1+\sqrt{5}}{2} \approx 0.69424$. They use a partitioning scheme that is simpler than the one described in this paper. Later it was observed by Edelsbrunner that a slight extension of the algorithm of [CSY] can be used to solve the following problem.

Given m points and n lines in the plane, find for each point the line that lies

vertically below it.

Since both problems are restricted cases of the problem studied in this paper, our algorithm yields new and more efficient solutions to these simpler problems as well. Note, however, that in these problems the output size is not a significant issue – the first problem is just a decision problem and the output in the second problem has only linear size. In contrast, for the problem studied in this paper the output size, and thus the space– and time–complexity of the algorithm, can be forced to be super-linear, that is, $\Omega(m^{2/3}n^{2/3})$. An obvious open problem that arises is whether the two simpler problems can be solved in $o(m^{2/3}n^{2/3})$ time. As will be shown elsewhere [EGSh], the straightforward generalizations of the two problems to three dimensions can indeed be solved faster than the problem of calculating the entire cells of the arrangement containing the given points.

Our approach to calculating faces in arrangements is based on a dualization of the problem which puts limits on its generality. Nevertheless, we could give an equivalent description of our technique using only the primal plane. We thus draw a random sample of r of the given points and then partition the lines into $O(r^2)$ so–called 3–*corridors* each being the primal equivalent of a triangle in the dual plane (see [HW]). Each point is passed to all 3–corridors that contain it. It is an interesting open problem whether or not this primal view of our technique can be generalized to apply to arrangements of other curves such as circles and alike.

Another open problem is to obtain a deterministic and efficient algorithm that will replace our randomized technique for computing faces in an arrangement of lines or line segments. The best deterministic solution we have uses a variant of the conjugation tree technique of [EW2], and takes time approximately on the order of $n^{\frac{2\alpha}{\alpha+1}} m^{\frac{1}{\alpha-1}}$ (see also [Ed, chapter 10]).

Acknowledgements

We wish to thank Ken Clarkson for providing valuable insights into the random sampling technique which have helped us in substantial improvements of our technique. Thanks are also extended to John Hershberger, Janos Pach, Raimund Seidel, Jack Snoeyink, Jorge Stolfi, and Emo Welzl for useful discussions on the problems studied in this paper.

References

[BO] Bentley, J. L. and Ottmann, T. A. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.* **C–28** (1979), 643–647.

[Ca] Canham, R. J. A theorem on arrangements of lines in the plane. *Israel J. Math.* **7** (1969), 393–397.

[CD] Chazelle, B. and Dobkin, D. P. Intersection of convex objects in two and three dimensions. *J. Assoc. Comput. Mach.* **34** (1987), 1–27.

[Cl] Clarkson, K. New applications of random sampling in computational geometry. *Discrete Comput. Geom.* **2** (1987), 195–222.

[CEGSW] Clarkson, K., Edelsbrunner, H., Guibas, L. J., Sharir, M. and Welzl, E. Combinatorial complexity bounds for arrangements. In preparation.

[CSY] Cole, R., Sharir M. and Yap, C. K. On k-hulls and related problems. *SIAM J. Comput.* **16** (1987), 61–77.

[Ed] Edelsbrunner, H. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, 1987.

[EGSh] Edelsbrunner, H., Guibas, L. J. and Sharir, M. The complexity of many cells in arrangements of planes and related problems. In preparation.

[EGSt] Edelsbrunner, H., Guibas, L. J. and Stolfi, J. Optimal point location in a monotone subdivision. *SIAM J. Comput.* **15** (1986), 317–340.

[EOS] Edelsbrunner, H., O'Rourke, J. and Seidel, R. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.* **15** (1986), 341–363.

[ES] Edelsbrunner, H. and Sharir, M. The maximum number of ways to stab n convex nonintersecting objects in the plane is $2n-2$. To appear in *Discrete Comput. Geom.*

[EW] Edelsbrunner, H. and Welzl, E. On the maximal number of edges of many faces in an arrangement. *J. Combin. Theory, Ser. A* **41** (1986), 159–166.

[EW2] Edelsbrunner, H. and Welzl, E. Halfplanar range search in linear space and $O(n^{0.695})$ query time. *Inform. Process. Lett.* **23** (1986), 289–293.

[Gr] Grünbaum, B. *Convex Polytopes.* John Wiley & Sons, London, 1967.

[GOS] Guibas, L. J., Overmars, M. H. and Sharir, M. Intersections, connectivity, and related problems for arrangements of line segments. In preparation.

[GSS] Guibas, L. J., Sharir, M. and Sifrony, S. On general motion planning problems with two degrees of freedom. To appear in "Proc. 4^{th} ACM Symp. Comput. Geom. 1988".

[HS] Hart, S. and Sharir, M. Nonlinearity of Davenport Schinzel sequences and of generalized path compression schemes. *Combinatorica* **6** (1986), 151–177.

[HW] Haussler, D. and Welzl, E. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.* **2** (1987), 127–151.

[Mo] Moise, E. E. *Geometric Topology in Dimension 2 and 3.* Springer-Verlag, New York, 1977.

[OR] O'Rourke, J. The signature of a plane curve. *SIAM J. Comput.* **15** (1986), 34–51.

[PSS] Pollack, R., Sharir, M. and Sifrony, S. Separating two simple polygons by a sequence of translations. *Discrete Comput. Geom.* **3** (1988), 123–136.

[PS] Preparata, F. P. and Shamos, M. I. *Computational Geometry - an Introduction.* Springer-Verlag, New York, 1985.

[SML] Schmitt, A., Müller, H. and Leister, W. Ray tracing algorithms - theory and practice. In *Theoretical Foundations of Computer Graphics and CAD* (R.A. Earnshaw, Ed.), NATO ASI Series, Vol. F40, Springer-Verlag, Berlin, 1988, 997–1030.

[ST] Szemerédi, E. and Trotter, W. T. Extremal problems in discrete geometry. *Combinatorica* **3** (1983), 381–392.

[WS] Wiernik, A. and Sharir, M. Planar realization of nonlinear Davenport Schinzel sequences by segments. *Discrete Comput. Geom.* **3** (1988), 15–47.

NYU COMPSCI TR-358 c.1
Edelsbrunner, Herbert
The complexity of many
faces in arrangements of
lines and of segments